Volume 2, Issue 2, February 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Query Optimization in Object Oriented Databases through Detecting Independent Subqueries

Ms. M.C. Nikose
Computer Science & Engineering
Sipna's College of Engg & Tech.
Amravati, Maharashtra.
India.

Ms. S.S. Dhande
Computer Science & Engineering
Sipna's College of Engg & Tech.
Amravati, Maharashtra.
India

Dr. G. R. Bamnote
Computer Science & Engineering
P.R.Meghe Institute of Tech. & Research
Badnera, Maharashtra.
India

Abstract - Query optimization is the refining process in database administration and it helps to bring down speed of execution. Some object-oriented languages allows to express queries explicitly in the code, which are optimized using the query optimization techniques from the database domain. With respect to this, a formalized object query language (OQL) has been developed that performs optimization of queries at compile time. Object Oriented Data Base (OODB) has all the features, functionality of a relational database system and also offers an Object Oriented Programming language interface. We follow the stack based approach to query languages, which is responsible for naming-scoping-binding principle. In this paper we proposed one of the methods of query optimization depending on rewriting. Optimization by rewriting concerns queries containing so called independent subqueries. It consists in detecting them and then factoring outside the loops implied by query operators.

Keywords: OQL, Object Oriented Database, Stack Based Approach, Rewriting, Query Optimization.

I. INTRODUCTION

During the last two decades, Relational Database Management System (RDBM) has been established as the technology, handling databases up to terabytes. Relational DBMSs have been extremely successful in the market; however RDBMS lack the mechanisms to deal with complex structured data. Their tabular approach does not allow a suitable modeling of complex hierarchical objects. Most of the applications such as Geographical Information System, CAD, Multimedia, and Engineering etc. are characterized by having to manage complex, highly interrelated information, which was difficult to manage in RDBMS. To combat the limitations of RDBMS and meet the challenge of the increasing rise of the internet and the Web, programmers developed object-oriented databases in 1980 [7].

In recent years, database research has concentrated on object-oriented data models, which allow to store highly structured data. With regard to the data structuring concepts offered, an object-oriented data model can be looked upon as an extension of the nested relational model, [5] which allows to store relations as attribute values. However, the relational model only permits the alphanumeric data management. A similar role in object-oriented database is fulfilled by object query languages (OQL). The usefulness of these languages strongly depends on query optimization. With growing complexity of data structuring concepts, the complexity of the accompanying query language grows as well and thus also the complexity of query processing and optimization.

For the correct and precise formalization of object query language the concept of naming-scoping-binding paradigm must take into consideration. Hence we fallow the Stack Based Approach (SBA). Analyzing query processing in the Stack Based Approach (SBA) [3], it can be observed that some subqueries are evaluated many times outside the loops implied by the non-algebraic operators despite that in subsequent loop cycles there results are same. Such subqueries can be processed only once and their result can be used in next loop cycles. This observation is a basis for our proposed method called factoring out independent subqueries. The underlying idea used here is that, if none of the name in subquery is bound in the ES section opened by the non-algebraic operator currently being evaluated, then this subquery is independent of this operator. This means that subquery can be factored out of that operator, i.e., executed outside its iteration loop.

II. OVERVIEW OF OODBMS

An OODBMS is the result of combining object oriented programming principles with database management principles. Object oriented programming concepts such as encapsulation, polymorphism and inheritance are enforced along with regular database management concepts such as the Atomicity, Consistency, Isolation and Durability (ACID properties) which lead to system integrity, support for an *ad hoc* query language and secondary storage management systems which allow for managing very large amounts of data.. OODB [6] is a system while supporting all the functionality of a relational database system (including queries, transactions, backup and recovery mechanisms), also offers an Object oriented programming language interface, user defined data types, object identifiers and the ability to manage objects persistently. Features that are common in the RDBMS world such as transactions, the ability to handle large amounts of data, indexes, deadlock detection, backup and restoration features and data recovery mechanisms also exist in the OODBMS world. Following figure shows the features of Object oriented Database [8]. A primary feature of an OODBMS is that accessing objects in the database is done in a transparent manner such that interaction with persistent objects is no different from interacting with in-memory objects.

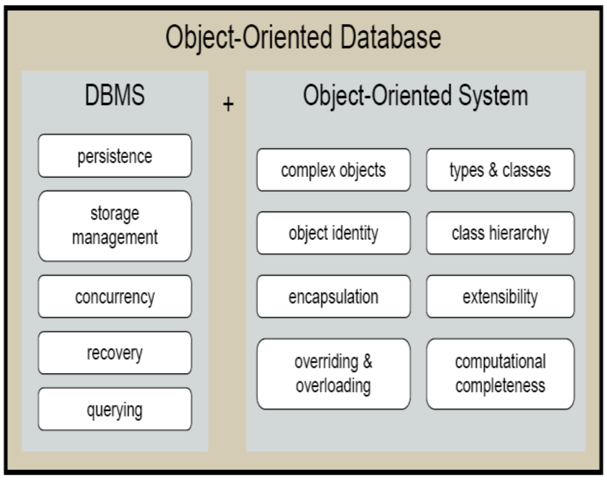

Fig 1. Features of OODBMS

Database operations typically involve obtaining a database root from the OODBMS which is usually a data structure like a graph, vector, hash table, or set and traversing it to obtain objects to create, update or delete from the database. When a client requests an object from the database, the object is transferred from the database into the application's cache where it can be used either as a transient value that is disconnected from its representation in the database or it can be used as a mirror of the version in the database in that updates to the object are reflected in the database and changes to object in the database require that the object is refetched from the OODBMS.

III. QUERY OPTIMIZATION IN OODBMS

Query optimization in relational databases is benefited a lot from the simplicity of the data model. This is not the case with the object model. The object-oriented data model is a generalization of the relational one and is believed to eliminate many of its flaws through incorporating modern concepts. Object models are descended of the semantic networks and object programming languages. They aim to permit the reuse of structures and operations to construct some more complex entities. [4] Data are represented in the basis as of objects. Associations are implemented by the direct ties via object identifying that permit a fast navigational access between the different objects. Indeed, a query must use the new concepts introduced by object model [3]. The class is a data abstract type permitting to define properties of a whole of objects regrouped in two categories: attributes and operations [5][6].The *object* is a triplet <OID, class, state>, the OID is identifying of the object. It's unique and invariant during the program. The *attribute* is defined by its name and its type. An *operation* is a function that permits to modify the state of an object or to send back a value. The *inheritance* is a transmission mechanism of properties of a class toward one under class. The *inheritance* is simple if the property is inherited of only one on-class. It is multiple when the property is present in several on-classes. The *polymorphism* is the fact to arrange operations having one same name but of the different parameters in number or in types.

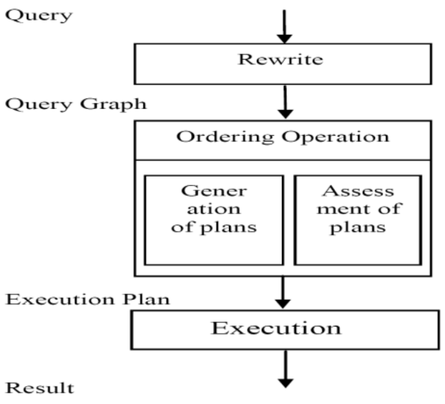

Fig 2. Optimization Process

Most of the query optimization methods are based on query rewriting [9]. Rewriting means transforming a query q1 into semantically equivalent query q2 promising much better performance. It consists in detecting parts of query matching some patterns. When it is recognized, a query is rewritten according to the predefined rewriting rule. The advantage of rewriting is that algorithm are fast, such optimization is compile time optimization entirely performed before query is executed, hence query optimization process itself does not burden the performance. Process of optimization is summarizes in three steps shown in figure 2 *Rewrite step* consists in a syntactic and semantic rewrite of the query in the goal to determine simpler equivalent queries [1]. The result of this step is the generation of a query graph. *Ordering operations step* is takes place in two phases: generation and assessment of the plans which determined in the first phase. *Execution step* permits to choose the optimal execution plan and to execute it

IV. STACK BASED APPROACH

The Stack-Based Approach (SBA) is a formal frame addressing object-oriented database query and programming languages (PLs). The approach is motivated by the belief that there is no definite border line between querying and programming; thus there should be a universal theory that uniformly covers both aspects. SBA allows [3] to precisely determine the semantics of query languages; there relation with object oriented concepts, with imperative programming constructs and with programming abstraction, including procedures, functional procedures views, modules etc. its main features are the following

- The naming-scoping-binding principle is assumed, which means that each name occurring in a query is bound to the appropriate run-time entity (an object, attribute, method, parameter, etc.) depending upon the scope for the name.
- One of its basic mechanisms is an environmental stack (ENVS). The stack is responsible for scope control, for binding names, parameter passing and procedure calls. ENVS is also responsible for processing non-algebraic query operators.
- The object relativity principle is assumed, i.e. object on any hierarchy level have same formal properties and are treated uniformly. This principle simplifies semantic considerations in developing query optimization methods.
- For objects the principle of internal identification is assumed; each run-time entity that can be separately bound, inserted, updated, deleted etc. must possess a unique internal identifier.

In all programming language the naming-scoping-binding issue leads to the mechanism, internal data structure called as Environmental Stack (ENVS)[1],[2]. It employs abstraction principle, which allows the programmer to consider the currently rewritten piece of code to be independent of the context of its possible use. Hence safe nesting of procedure calls is assured, including any recursive calls. ENVS is subdivided into sections, which are ordered, with newest section known as top and oldest one as bottom. A section is associated with a particular *procedure call* or an executed *program block.*

The stack consists of sections that are sets of binders. Binder is a concept that allows us to explain and describe various naming issues that occur in object models and programming languages. At the beginning, the ENVS consist of single section containing binders to all root database objects. During query evaluation the stack is growing and shrinking according to query nesting. Here we assume a small database schema shown below in fig 3.It defines five classes as Person, Professor, Student, Lecture and Faculty. Names of classes are followed by cardinality numbers, unless the cardinality is [1….1].Queries are combined by operators. All the operators use for joining a query are either unary or binary. Binary operator is subdivided into algebraic and non-algebraic. The main difference between them is that whether they modify the state of ENVS during evaluation or not.

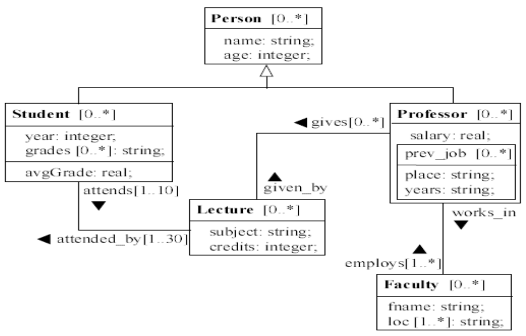

Fig. 3 Database Schema

An operator is algebraic if it does not modify the state of ENVS. The algebraic operators include numerical and string operators and comparisons, Boolean and, or, not, aggregate function, and sequence operators and comparisons, structure constructor, etc. Operators which name a query result are unary algebraic operators too. The operator *group as* names the entire query result, while *as* names each element in a sequence or bag returned by the query. If q1 Δ q2 be a query consisting of two subqueries connected by a binary algebraic operator Δ. The eval procedure takes q1 and pushes its result onto top of QERS then does the same with q2 , performs Δ with two top QRES values and finally removes top of QRES twice and pushes the final result onto top.

If query q1 Ө q2 involves a non-algebraic Ө, then q2 is evaluated in the context of q1. The context is determined by the new section opened by the Ө operator on ENVS for an element of q1. A new stack section pushed onto ENVS is constructed by a special function. Subqueries q1 and q2 cannot be processed independently, the order of evaluation is important. Non-algebraic operators include projection /navigation (q1.q2), selection (q1 where q2), dependent join (q1 join q2), quantifiers (Ǝq1q2), transitive closure and ordering.

V. BASIC IDEA OF PRPOSED METHOD

In this paper we are proposing one of the methods of query optimization. The idea of this method is based upon the observation that, if none of the name in subquery is bound in the ENVS section opened by the non-algebraic operator currently being evaluated, then this subquery is independent of this operator. Subqueries are called independent if they can be evaluated outside loops implied by the non-algebraic query operators. Such subqueries are worth analyzing because they usually imply optimization possibilities [1],[5].We use a special technique called the method of independent subqueries to optimize queries.

Technically, it consists in analyzing in which sections particular names occurring in a query are bound. It turns out that if none of the names in given subquery is bound in the scope opened by the non-algebraic operator currently being evaluated, then that subquery can be evaluated earlier than it results from its textual place in the query it is a part of. The method modifies the textual form of a query so that all its subqueries will be evaluated as soon as possible. To determine in which scopes names occurring in a query are bound at run time [10], we statically analyze that query and during analyzing we additionally do the following:

- Each non-algebraic operator is assigned the number of the scope it opens,
- Each name in the query is assigned two numbers:
  - The stack size: the number of scopes thst is on static ENVS when the binding of this name is being performed.
  - The binding level: the number of the scope on the stack in which this name is bound.

All those numbers are determined relatively to the bottom scopes of a query. Thus, for instance the binding level for a free name (names that are bound in the bottom scopes) is 1, and for non-free name (names that are not free) is greater than 1. The independent subquery method consists in analyzing the number of those ENVS section in which particular names occurring in a query are bound.

For example, consider a simple query "Get lectures whose credits are greater than the credits of physics"

Lecture where credits >
((Lecture where subject =”physics”).credits) (1)

Here the subquery returning the credits of physics:

(Lecture where subject =”physics”).credits (2)

Is evaluated for each Lecture object existing in the database, while it is enough to calculate it just once, because its evaluation gives the same result every time. We can say that this subquery is independent of its direct non-algebraic operator. Let us see how the number are assigned to query,

```
Lecture where credits >
(1, 1)    2    (2,2)
     ((Lecture where subject =”physics”).credits)
        (2,1)   3    (3,3)        3  (3,3)
```

As shown above none of the names in subquery (2) is bound in section 2 opened by external where, therefore subquery (2) is independent of that operator and can be calculated before it opens its section. To express it in the textual form of the query, the independent subquery is factored out this is done in following ways,

- A new unique auxiliary name is chosen. It will be used as the name of the result of the independent subquery (2).
- Then the subquery (2) is named by the *as* operator, put before the entire subquery (1) of the non-algebraic operator it is independent of, and connected to the rest of the query (i.e. 3) by a dot operator.
- Finally, the auxiliary name is put in the previous place of (2) this subquery. After factoring subquery (2) out, query (1) will be rewrite as

```
(((Lecture where subject =”physics”).credits) as c).
   (1,1)    2    (2,2)              2  (2,2)    2
      Lecture where credits > c                    (3)
        (2,1)   3    (3,3)   (3,2)
```

Where c is the auxiliary name. Now subquery (2) is evaluated before its result is used and the auxiliary name c naming its result makes it possible to read this result. The method of independent subquery is quite sophisticated , it recursively traversed a query abstract syntax tree(AST) to find the largest subquery which is independent of the currently evaluated non-algebraic operator. After detecting such subquery AST is reorganized according to the rewriting rule. The process is repeated until all independent subqueries are discovered and rewritten.

VI. CONCLUSION

The necessity to support complex data in databases is intensified. Models trying to answer to these needs appeared as the object-oriented and the object relational model. The relational model only permits the alphanumeric data management. To combat the limitations of RDBMS and meet the challenge of the increasing rise of the internet and the Web, programmers developed object-oriented databases. In this paper we discuss some basic concepts and features of OODBMS. We present a new optimization method for queries involving independent subquery. More powerful variants of the method were received on the assumption concerning the distributive property of selection, projection/navigation and dependent join. Due this property we can develop extended version of query rewriting methods known from the relational model, in particular, pushing selection before a join.

REFERENCES

[1]**[JPlod00]** J. Płodzień, A. Kraken, “Object Query Optimization through Detecting Independent Subqueries”, *Information Systems*, Elsevier Science, 25(8), 2000, pp. 467-490.

[2]**[Mich09]** Michel Bleja, Krzysztof Stencel, Kazimierz Subeita, “Optimization of Object-Oriented Queries Addressing Large and Small Collections”, Proc. Of the IMCSIT, 2009, ISBN 978-83-60810-22-4, Vol. 4, pp. 643-680.

[3]**[Subi95]** K.Subieta, C.Beeri, F.Matthes, J.W.Schmidt. “*A Stack-Based Approach to Query Languages”*. Proc.2nd East-West Database Workshop, 1994, Springer Workshops in Computing, 1995, 159-180.

[4]**[MA05]**Minyar Sassi, and Amel Grissa-Touzi “Contribution to the Query Optimization in the Object-Oriented Databases” World Academy of Science, Engineering and Technology 11 2005

[5][G99]G. Gardarin, “*Object and relational databases*” , Eyrolles, 1999.

[6] [RC 94]R.G.G. Catell, “*Object-Oriented Data Management: Object-Oriented and Extended Relational Database Sytems*”, Addison-Wesley Publishing, Inc., 1994.

[7] Sunanda Luthra “*Architecture In Object Oriented databases* ”Lecturer, Department of CSE/IT Amritsar College of Engg. & Tech, Amritsar.143001,*Punjab, India* .

[8] David Maier “Object-Oriented Database Theory An Introduction & Indexing in OODBS”

[9][Plod00] J. Plodzien, “Optimization Methods in Object query Languages”, Ph. D. Thesis, Institute of Computer Science, Polish Academy of Sciences, 2000.